H

**COMP1411 (Spring 2021) Introduction to Computer Systems**

Individual Assignment 3 Due Date: 10:00am, 28th April, 2021

|  |  |
| --- | --- |
| *Name* |  |
| *Student number* |  |

**Question 1** [4 marks]

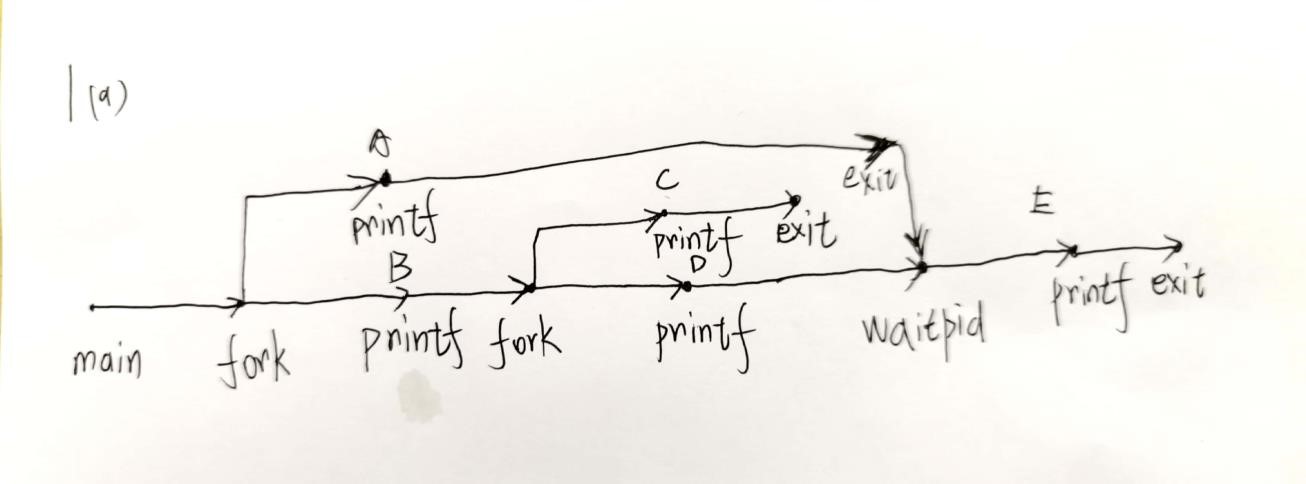
Consider the execution of the following function (written in the C language).

// all the needed headers are included int main() { int pid; if ((pid = fork()) == 0){ printf("A"); fflush(stdout); exit(1); } printf("B"); fflush(stdout); if (fork() == 0){ printf("C"); fflush(stdout); exit(2); } printf("D"); fflush(stdout); waitpid(pid, NULL, 0); printf("E"); fflush(stdout); exit(0); }

**1(a)** **Draw** the process graph for the execution of “main()”.

In a process graph, each function, including main(), fork(), printf(), waitpid(), and exit(), is represented by a vertex. For each vertex, please write the function name below the vertex and for printf() write the output character above the vertex. Each edge must be directed, with the direction representing the happen before relationship.

*Answer:*



**1(b)** **List** all the feasible output of “main()”.

For example, if there are two feasible outputs, please given the answer as follows (listing only one feasible output in each line and number them):

1. A B C D E
2. E D C B A

Note that the above two outputs are only used to demonstrate the format to give your answers for question 1(b), they do not have any indication to the correct answers.

*Answer:*

(1)ABCDE (5)BADCE (9)BDCAE

(2)ABDCE (6)BADEC (10)BCDAE

(3)ABDEC (7)BCADE (11)BDAEC

(4)BACDE (8)BDACE

**Question 2** [4 marks]

Considering a virtual memory system with paging:

Assume that the **page size** is **2KB (B = bytes)**, and each **physical and virtual address** is represented by a **16-bit binary** or equivalently a **4-digit hex-decimal** number.

A process was assigned **3 physical pages** with physical page number 2, 4, 7. This means that no matter how many virtual pages the process has, all its virtual pages must be mapped into only the 3 given physical pages.

Now it is **time** 130 (a larger time value indicates a later point in time). The current content of the page table is shown below with Table 1, in which the “**Access time**” column shows the **time of the most recent visit to the virtual page** in the corresponding row (e.g, 120 indicates that virtual page 3 is visited most recently at time 120).

Page numbers are given in the **decimal format**. The “-” symbol in the virtual page column indicates that no virtual page has been mapped to the physical page that are listed in the same row with the virtual page. The “-” symbol in the “Access time” column indicates that this physical page is not visited yet.

Assume that **LRU replacement** is used for page replacement. This means, if all the 3 physical pages have been mapped with 3 virtual pages, and a 4th different virtual page is accessed, among the 3 mapped virtual pages, the one with the **oldest (smallest) “Access time”** will be replaced by the 4th new virtual page.

From now on, the following virtual addresses will be accessed sequentially:

**0x1D8C**, **0x4397**, **0x37D5**, **0x2A4F** at time **150**, **160**, **200** and **300** respectively.

**2(a)** Please **translate** each virtual address into the correct physical address. Note that each physical address should be represented in the hex-decimal format, with 0x as the prefix. **2(b)** Please **complete** the page table after the above four accesses, by filling in the blanks in Table 2. Note that you are required to input the page numbers in decimal format.

**Table 1: The original page table**

|  |  |  |
| --- | --- | --- |
| **Virtual page number** | **Physical page number** | **Access time** |
| 3 | 2 | 120 |
| 5 | 4 | 80 |
| - | 7 | - |

*Answer:*

2(a)

0x1D8C → 0x158C

0x4397 → 0x3B97

0x37D5 → 0x27D5

0x2A4F → 0x124F

2(b)

**Table 2**

|  |  |  |
| --- | --- | --- |
| **Virtual page number** | **Physical page number** | **Access time** |
| \_\_\_\_5\_\_\_\_ | 2 | \_\_\_\_300\_\_\_ |
| \_\_\_6\_\_\_ | 4 | \_\_\_200\_\_\_ |
| \_\_\_\_8\_\_\_\_ | 7 | \_\_\_\_160\_\_\_\_ |

**Question 3** [2 marks]

Assume the system has a cache between the CPU and the main memory. Each time the CPU uses some data item, the CPU will always go the cache to fetch the data item; if the data item is found in the cache, then there is a cache hit, and the data item will be loaded to CPU; otherwise if the data item is not found in the cache, then the item will be loaded from the main memory and at the same time put into the cache (we assume that all the access data are guaranteed to be stored in the main memory).

Each **cache block** has the size of **16B** (B = bytes), and **the cache has 3 blocks**, which means the **total size** of the cache is **48B**. If one main memory address is accessed, the whole block containing the accessed address will be loaded into the cache.

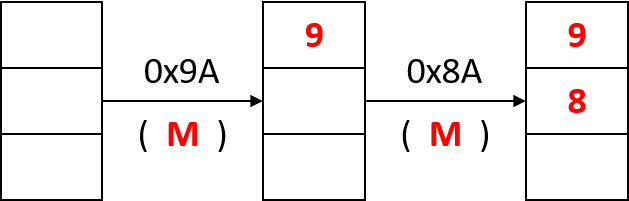
We assume that addresses in 0x00-0x0F belong to a data block with ID 0, addresses in 0x100x1F belong to a data block with ID 1, addresses in 0x20-0x2F belong to a data block with ID 2, addresses in 0x30-0x3F belong to a data block with ID 3.

The cache is managed with the **FIFO** replacement policy: when all the 3 cache blocks are used up and a new data block will be loaded into cache, one data block out of the 3 blocks will be replaced out of the cache. The data block that was **earliest loaded** into the cache will be replaced. We assume that at the very beginning, the cache is **empty**.

**10 main memory addresses** are accessed sequentially, listed in Figure 2.

**Please fill in Figure 2**: input the corresponding data block ID for each accessed address into the cache states represented by vertical boxes, and input whether the access to the address is cache hit or miss by filling the brackets below the address, with “M” for cache miss and “H” for cache hit.

For example, starting from empty cache, access address 0x9A and 0x8A, the red contents are those you are supposed to fill in, as shown in Figure 1.



**Figure 1**

*Answer:*

**Figure 2**

